Docket No.: 0023-0059

MULTIPLE CONCURRENT DEQUEUE ARBITERS

BACKGROUND OF THE INVENTION

A. Field of the Invention

[0001] The present invention relates generally to arbitration, and more particularly, to a high performance dequeuing arbitration scheme.

B. Description of Related Art

[0002] Routers receive data on a physical media, such as optical fiber, analyze the data to determine its destination, and output the data on a physical media in accordance with the destination. Routers were initially designed using a general purpose processor executing large software programs. As line rates and traffic volume increased, however, general purpose processors could not scale to meet these new demands. For example, as functionality was added to the software, such as accounting and policing functionality, these routers suffered performance degradation. In some instances, the routers failed to handle traffic at the required line rate when the new functionality was enabled.

[0003] To meet the new demands, purpose-built routers were designed. Purpose-built routers are designed and built with components optimized for routing. They not only handle higher line rates and higher network traffic volume, but they also add functionality without compromising line rate performance.

[0004] A purpose-built router may include a number of input and output ports from which it transmits and receives information packets. A switching fabric may be implemented in the router to carry the packets between ports.

[0005] In order to control their high packet throughput, purpose-built routers use buffers to temporarily queue packets waiting to be processed. Arbiters may control the dequeuing of packets from the buffers. Different arbiters may operate on the same buffer to control different aspects of the buffering and dequeuing process. For example, one arbiter may select packets from the queues for transmission while another arbiter may examine the queues for congestion and drop packets from congested queues.

[0006] When using multiple arbiters that arbitrate over the same set of queues, it is desirable to implement the arbiters in a manner that is as efficient as possible. Preferably, total bandwidth through the arbiters should be maximized while sharing common resources related to the buffers.

SUMMARY OF THE INVENTION

[0007] Multiple arbiters share common resources of a number of queues. Conflict detection logic allows the arbiters to operate at a high combined bandwidth while giving preference to certain of the arbiters.

[0008] More specifically, in one aspect, concepts consistent with the invention include a system including arbiters that arbitrate among elements of a common resource. The system additionally includes conflict logic configured to detect conflicts among the elements of the common resource. When a conflict is detected, the conflict logic alters processing relating to the conflict in one of the conflicting arbiters.

[0009] Another aspect consistent with the invention is directed to a method having a number of acts. The acts include examining arbiters that arbitrate among queues for conflicts in arbitrating the queues and determining, when conflicts occur in arbitrating the queues, whether one of the conflicting arbiters has reached an arbitration point beyond a predetermined commit point. Additionally, the method includes invalidating processing in one arbiter related to the conflict when the one arbiter is not beyond the commit point.

[0010] Yet another aspect consistent with the principles of the invention is directed to a device including a number of queues and first and second arbiters. The first arbiter is configured to select from among the queues and to receive data items from the selected queue. The second arbiter is configured to monitor the queues for congestion and to drop data items from congested queues. Additionally, conflict detection logic detects conflicts between the first and second arbiters in arbitrating the queues. When a conflict is detected, the logic alters processing relating to the conflict in the one of the arbiters when the arbiter has not passed a predetermined commit point in processing.

[0011] Yet another aspect consistent with the principles of the invention is directed to a network device comprising processing elements that transmit data items to one another and transmit the data items to destinations external to the network device. The processing elements include queues configured to store the data items before transmission of the data items, arbiters that independently arbitrate among data items in the queues, and conflict logic. The conflict logic detects conflicts among the arbiters in accessing the queues, and, when a

conflict is detected, the conflict logic clears processing relating to the conflict in one of the conflicting arbiters when the one of the conflicting arbiters has not passed a predetermined commit point.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate an embodiment of the invention and, together with the description, explain the invention. In the drawings,

[0013] Fig. 1 is a block diagram illustrating an exemplary routing system in which systems and methods consistent with the principles of the invention may be implemented;

[0014] Fig. 2 is a detailed block diagram illustrating portions of the routing system shown in Fig. 1;

[0015] Fig. 3 is a diagram conceptually illustrating notification flow through queues;

[0016] Fig. 4 is a diagram illustrating a parallel implementation of arbiters consistent with principles of the invention; and

[0017] Fig. 5 is a flow chart illustrating the operation of the arbiters of Fig. 4.

DETAILED DESCRIPTION

[0018] The following detailed description of the invention refers to the accompanying drawings. The same reference numbers in different drawings may identify the same or similar elements. Also, the following detailed description does not limit the invention. Instead, the scope of the invention is defined by the appended claims and equivalents.

[0019] As described herein, a first arbiter arbitrates over a set of queues. A second arbiter independently arbitrates over the same set of queues. Conflict detection logic prioritizes the arbiters while maximizing total bandwidth of the two arbiters.

SYSTEM OVERVIEW

[0020] Fig. 1 is a block diagram illustrating an exemplary routing system 100 in which the present invention may be implemented. System 100 receives a data stream from a physical link, processes the data stream to determine destination information, and transmits the data stream out on a link in accordance with the destination information. System 100 may include packet forwarding engines (PFEs) 104, a switch fabric 110, and a routing engine (RE) 102.

[0021] RE 102 performs high level management functions for system 100. For example, RE 102 communicates with other networks and systems connected to system 100 to exchange information regarding network topology. RE 102 creates routing tables based on network topology information, creates forwarding tables based on the routing tables, and forwards the forwarding tables to PFEs

104. PFEs 104 use the forwarding tables to perform route lookup for incoming packets. RE 102 also performs other general control and monitoring functions for system 100.

[0022] PFEs 104 are each connected to RE 102 and switch fabric 110. PFEs 104 receive data at ports on physical links connected to a network, such as a wide area network (WAN). Each physical link could be one of many types of transport media, such as optical fiber or Ethernet cable. The data on the physical link is formatted according to one of several protocols, such as the synchronous optical network (SONET) standard or Ethernet.

[0023] PFEs 104 process incoming data by stripping off the data link layer. PFEs 104 convert the remaining data into data structures referred to herein as D cells (where a cell may be a fixed length data unit). For example, in one embodiment, the data remaining after the data link layer is stripped off is packets. PFE 104 includes layer 2 (L2) and layer 3 (L3) packet header information, some control information regarding the packets, and the packet payload data in a series of D cells. In one embodiment, the L2, L3, and the control information are stored in the first two cells of the series of cells. The packet's payload data may also be stored as a series of cells.

[0024] PFEs 104 form data structures called notifications based on the L2, L3, and control information, and perform route lookups using the notification and the routing table from RE 102 to determine destination information. PFEs 104 may also further process the notification to perform protocol-specific

functions, policing, and accounting, and might even modify the notification to form a new notification.

[0025] If the determined destination indicates that the packet should be sent out on a physical link connected to one of PFEs 104, then PFE 104 retrieves the cells for the packet, converts the notification or new notification into header information, forms a packet using the packet payload data from the cells and the header information, and transmits the packet from the port associated with the physical link.

[0026] If the destination indicates that the packet should be sent to another PFE via switch fabric 110, then the PFE 104 retrieves the cells for the packet, modifies the first two cells with the new notification and new control information, if necessary, and sends the cells to the other PFE via switch fabric 110. Before transmitting the cells over switch fabric 110, PFE 104 appends a sequence number to each cell, which allows the receiving PFE to reconstruct the order of the transmitted cells. Additionally, the receiving PFE uses the notification to form a packet using the packet data from the cells, and sends the packet out on the port associated with the appropriate physical link of the receiving PFE.

[0027] In summary, in one embodiment, RE 102, PFEs 104, and switch fabric 110 perform routing based on packet-level processing. PFEs 104 store each packet in cells while performing a route lookup using a notification, which is based on packet header information. A packet might be received on one PFE and go back out to the network on the same PFE, or be sent through switch fabric 110 to be sent out to the network on a different PFE.

[0028] Fig. 2 is an exemplary detailed block diagram illustrating portions of routing system 100. PFEs 104 connect to one another through switch fabric 110. Each of PFEs 104 may include one or more physical interface cards (PICs) 210 and flexible port concentrators (FPCs) 220.

[0029] PICs 210 may transmit data between a WAN physical link and FPC 220. Different PICs are designed to handle different types of WAN physical links. For example, one of PICs 210 may be an interface for an optical link while the other PIC may be an interface for an Ethernet link.

[0030] For incoming data, in one embodiment, PICs 210 may strip off the layer 1 (L1) protocol information and forward the remaining data, such as raw packets, to FPC 220. For outgoing data, PICs 210 may receive packets from FPC 220, encapsulate the packets in L1 protocol information, and transmit the data on the physical WAN link.

[0031] FPCs 220 perform routing functions and handle packet transfers to and from PICs 210 and switch fabric 110. For each packet it handles, FPC 220 may perform the previously-discussed route lookup function. Although Fig. 2 shows two PICs 210 connected to each of FPCs 220 and three FPCs 220 connected to switch fabric 110, in other embodiments consistent with principles of the invention there can be more or fewer PICs 210 and FPCs 220.

ARBITRATION OVERVIEW

[0032] As noted above, FPCs 220 generate notifications for received packets. The notifications may include a reference to the actual packet data

stored in memory and the appropriate outgoing interface (i.e., an outgoing port on one of PICs 210) associated with the packet. The notifications may then stored in queues corresponding to the outgoing interface. For example, the notifications may be placed in one of a number of dedicated first-in-first-out (FIFO) queues. The FIFO queues may be prioritized so that higher priority packets have their notifications sent to higher priority queues.

[0033] Fig. 3 is a diagram conceptually illustrating notification data flow through a number of queues 301-303. A notification that reaches the head position in its queue 301-303 may be selected by arbiter 310. Notifications selected by arbiter 310 may be used to retrieve their corresponding packet data before being transmitted from system 100.

[0034] In Fig. 3, notifications selected by arbiter 310 for a particular group of queues are assembled into a stream 320. Typically, a stream 320 may correspond to a particular output port on one of PICs 210. Each queue accordingly shares the bandwidth of the stream 320. Arbiter 310 may allow higher priority ones of queues 301-303 to use a greater portion of the bandwidth of stream 320 than lower priority queues. In this manner, arbiter 310 may control the flow of packets from its input queues. This type of arbitration, in which packets are selected based on flow control concerns related to the bandwidth of stream 320 will be referred to herein as "DQ" arbitration.

[0035] In addition to managing the flow of notifications from queues 301-303 based on queue priority, arbiter 310 may manage queue congestion by dropping notifications from one or more queues according to a probability that

increases as the latency through one or more queues increases. In other words, when managing congestion in a queue, arbiters 310 may drop entries, on a per-queue basis, as the queues become congested. One known technique for probabilistically dropping data items from a queue based on congestion is known as a Random Early Drop (RED) process. In general, RED algorithms are well known in the art and therefore will not be described further herein.

[0036] To maximize arbitration efficiency, it is desirable for arbiter 310 to simultaneously implement both DQ arbitration and RED arbitration on the same set of queues.

PARALLEL ARBITRATION IMPLEMENTATION

[0037] Fig. 4 is a diagram illustrating a parallel implementation of RED and DQ arbitration schemes consistent with principles of the invention. Arbitration system 400 includes a DQ arbiter 401 and a RED arbiter 402 that operate on queue component 410. Queue component 410 includes a series of queues 421-423, such as FIFO queues. Queues 421-423 may correspond, for example, to different packet priority transmission levels that store notifications corresponding to the packets. Queues 421-423 may each be associated with corresponding local queue control logic (QCL) 431-433. Local QCL 431-433 handles the details associated with enqueuing and dequeuing data items from its associated queue. Queue component 410 additionally includes a set of shared queue resources 440 and common control logic 445. Shared queue resources 440 include, for example, memory pointer registers for each queue that store the current head

(next data item in the queue) and tail (last, or most recently added data item in the queue) locations in the queue, and bit vectors used to indicate whether a queue is busy (e.g., being accessed). Similarly, common control logic 445 provides common control functionality for queues 421-423.

[0038] DQ arbiter 401 and RED arbiter 402 may each be implemented as a series of pipelined stages. DQ arbiter 401 and RED arbiter 402 may each include of a different number of stages. In one implementation, DQ arbiter 401 may be an eight stage pipeline and RED arbiter 402 may be a fourteen stage pipeline. Both RED arbiter 402 and DQ arbiter 401 may select a new queue every two cycles. The pipelines may be structured so that the early stages of the DQ and RED pipelines read data from queues 421-423 and the later stages of the pipeline write back or update the queue head data pointers in shared resources 440.

[0039] DQ arbiter 401 and RED arbiter 402 independently access queues 421-423, and their corresponding resources, in queue component 410. Kill logic 403 provides conflict detection between DQ arbiter 401 and RED arbiter 402. When DQ arbiter 401 and RED arbiter 402 attempt to access the same one of queues 421-423, kill logic 403 halts the access by one of DQ arbiter 401 or RED arbiter 402 when the kill logic 403 detects that the multiple accesses will lead to an error. For example, in one implementation, if DQ logic 401 attempts to access a queue that is already being accessed by RED arbiter 402, kill logic 403 will stop the access by RED arbiter 402 as long as RED arbiter 402 has not progressed beyond a predetermined "commit" point in its pipeline. The commit point is the

stage in the RED arbiter's pipeline that starts to write to or modify one of queues 421-423. Thus, if stages one through eight of the pipeline of RED arbiter 402 are read stages and stage nine begins a write stage back to the active queue 421-423, kill logic 403 may kill the queue access by RED arbiter 402 up until stage nine. In this example, kill logic 403 generally attempts to give priority to DQ arbiter 401.

[0040] Fig. 5 is a flow chart illustrating the operation of arbitration system 400. In general, RED arbiter 402 and DQ arbiter 401 operate independently of one another on queues 421-423, and thus each independently select their next queue on which to operate (act 501). Kill logic 403 and common control logic 445 receive each arbiter's next active queue selection (act 502). For example, each arbiter may transmit a queue number, to queue component 410, indicating its current selection. In response, common control logic 445 begins to transmit queue data, such as the next data item from the selected queue or an indication of whether the selected queue is busy. Additionally, control logic 445 may begin to update shared resources 440 by, for example, setting a bit to indicate that the selected queue is now busy (act 511).

[0041] Concurrently with act 511, kill logic 403 examines the selected queues for possible resource conflicts (act 503). A conflict may occur if DQ arbiter 401 attempts to access a queue while RED arbiter 402 has already started a queue access (or vice-versa). If a conflict is detected, kill logic 403 determines the processing state of the queue by RED arbiter 402 to determine if it is beyond its commit state (acts 504 and 505). If RED arbiter 402 is not beyond

its commit stage, kill logic 403 invalidates the entries in the pipeline stages in RED arbiter 402 that relate to the conflict (act 506). If RED arbiter 402 is beyond its commit stage, it is too late to cancel the RED arbiter's queue access. In this situation, common control logic 445 may still allow DQ arbiter 401 to continue operation. More particularly, common control logic 445 may advance the queue head pointer in shared resource component 440 to its next logical position before sending the queue's data item to DQ arbiter 401. In this manner, DQ arbiter 401 bypasses the normal queue head pointer and uses the next position of the head pointer when accessing the queue. Because RED arbiter 402 operates to drop data items from queues 421-423, and does not care about the substantive contents of queues 421-423, this type of "bypass" operation does not impact DQ arbiter 401. Accordingly, if a bypass operation is possible (i.e., the selected queue contains at least one additional data item) and RED arbiter 402 decides to drop its data item, common control logic 445 bypasses the next data item in queues 421-423 and advances the position of the queue's head pointer to the following entry in the queue (acts 507, 508, and 511). If a bypass operation is possible but RED arbiter 402 decides not to drop its data item, common control logic 445 allows DQ arbiter 401 to continue normal operation (acts 507, 508, 510). Otherwise, if the bypass operation is not possible, kill logic 403 invalidates the entries in the DQ arbiter's pipeline that relate to the selected queue (acts 507 and 509). RED arbiter 402 may continue to work on its selected queue (act 510).

SUMMARY

[0042] The arbitration scheme described herein provides for a number of desirable features. One of these features is that per queue, the arbitration scheme allows both DQ and RED arbitration schemes to run such that the DQ arbitration is not affected by the RED arbitration while allowing the RED arbitration to fully use all remaining bandwidth. Additionally, when aggregated across all queues, the arbitration scheme tends to maximize total RED and DQ bandwidth. Further, the arbitration scheme prevents any systematic bias for or against RED arbitration based on DQ arbitration activity and minimizes port and hardware implementation space needed to share resources used by the DQ and RED arbitration.

[0043] Although the above descriptions have been in the context of a DQ arbiter and a RED arbiter, the concepts consistent with the invention are not limited to these two types of arbiters. Other and additional numbers of arbiters could be used in their place.

[0044] It will be apparent to one of ordinary skill in the art that the embodiments as described above may be implemented in many different forms of software, firmware, and hardware in the entities illustrated in the figures. The actual specialized control hardware used to implement aspects consistent with principles of the invention is not limiting of the present invention.

[0045] The foregoing description of preferred embodiments of the present invention provides illustration and description, but is not intended to be exhaustive or to limit the invention to the precise form disclosed. Modifications

and variations are possible in light of the above teachings or may be acquired from practice of the invention.

[0046] No element, act, or instruction used in the description of the present application should be construed as critical or essential to the invention unless explicitly described as such. Also, as used herein, the article “a” is intended to include one or more items. Where only one item is intended, the term “one” or similar language is used.

[0047] The scope of the invention is defined by the claims and their equivalents.